图文 114 RocketMQ是如何实现同步刷盘以及异步刷盘两种策略的?

详情 评论

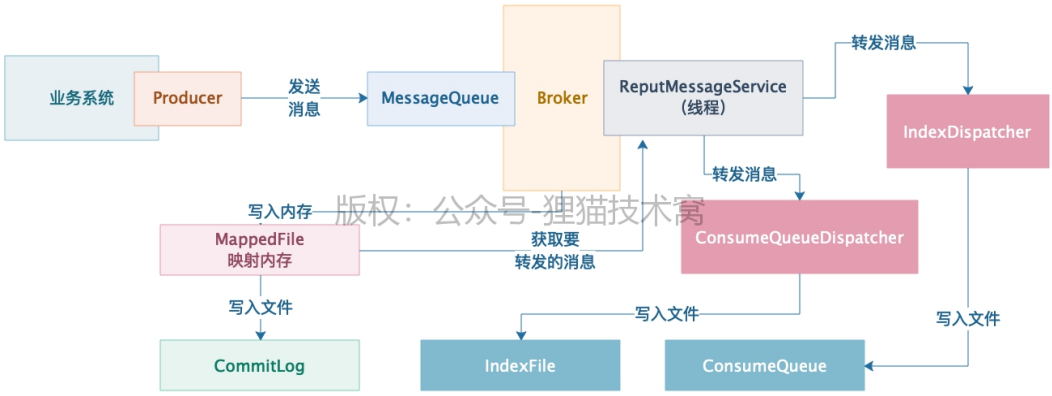
RocketMQ是如何实现同步刷盘以及异步刷盘两种策略的?

继《从零开始带你成为JVM实战高手》后，阿里资深技术专家携新作再度出山，重磅推荐：

（点击下方蓝字试听）

《从零开始带你成为MySQL实战优化高手》

上一次我们已经给大家讲解完了数据写入到Broker之后的存储流程，包括数据直接写入CommitLog，而且直接进入的是MappedFile映射的一块内存，不是直接进入磁盘，同时有一个后台线程会把CommitLog里更新的数据给写入到ConsumeQueue和IndexFile里去，如下图所示



我们之前简单提过一次，写入CommitLog的数据进入到MappedFile映射的一块内存里之后，后续会执行刷盘策略

比如是同步刷盘还是异步刷盘，如果是同步刷盘，那么此时就会直接把内存里的数据写入磁盘文件，如果是异步刷盘，那么就是过一段时间之后，再把数据刷入磁盘文件里去。

那么今天我们来看看底层到底是如何执行不同的刷盘策略的。

大家应该还记得之前我们说过，往CommitLog里写数据的时候，是调用的CommitLog类的putMessage()这个方法吧?

没错的，其实在这个方法的末尾有两行代码，很关键的，大家看一下下面的源码片段。

```
public PutMessageResult putMessage(final MessageExtBrokerInner msg) {
  // 省略了一大段源码
  handleDiskFlush(result, putMessageResult, msg);
  handleHA(result, putMessageResult, msg);
  return putMessageResult;
}
```

大家会发现在末尾有两个方法调用，一个是handleDishFlush()，一个是handleHA()

顾名思义，一个就是用于决定如何进行刷盘的，一个是用于决定如何把消息同步给Slave Broker的。

关于消息如何同步给Slave Broker，这个我们就不看了，因为涉及到Broker高可用机制，这里展开说就太多了，其实大家有兴趣可以自己慢慢去研究，我们这里主要就是讲解一些RocketMQ的核心源码原理。

所以我们重点进入到handleDiskFlush()方法里去，看看他是如何处理刷盘的。

```
public void handleDiskFlush(
        AppendMessageResult result,
        PutMessageResult putMessageResult,
        MessageExt messageExt) {
  // Synchronization flush
  if (FlushDiskType.SYNC_FLUSH == this.defaultMessageStore
        .getMessageStoreConfig().getFlushDiskType()) {
      // 省略一大片代码
  }
  // Asynchronous flush
  else {
     // 省略一大片代码
  }
}
```

上面代码我们就看的很清晰了，其实他里面是根据你配置的两种不同的刷盘策略分别处理的，我们先看第一种，就是同步刷盘的策略是如何处理的。

```
final GroupCommitService service =
    (GroupCommitService) this.flushCommitLogService;

if (messageExt.isWaitStoreMsgOK()) {
    GroupCommitRequest request = new GroupCommitRequest(
        result.getWroteOffset() + result.getWroteBytes());

    service.putRequest(request);
    boolean flushOK = request.waitForFlush(
                        this.defaultMessageStore.
                        getMessageStoreConfig().
                        getSyncFlushTimeout());
    if (!flushOK) {
        log.error(
            "do groupcommit, wait for flush failed, topic: " +
            messageExt.getTopic() +
            " tags: " +
            messageExt.getTags() +
            " client address: " +
            messageExt.getBornHostString());

        putMessageResult.setPutMessageStatus(
            PutMessageStatus.FLUSH_DISK_TIMEOUT);
    }
} else {
    service.wakeup();
}
```

其实上面就是构建了一个GroupCommitRequest，然后提交给了GroupCommitService去进行处理，然后调用request.waitForFlush()方法等待同步刷盘成功

万一刷盘失败了，就打印日志。具体刷盘是由GroupCommitService执行的，他的doCommit()方法最终会执行同步刷盘的逻辑，里面有如下代码。

```
CommitLog.this.mappedFileQueue.flush(0);
```

上面那行代码一层一层调用下去，最终刷盘其实是靠的MappedByteBuffer的force()方法，如下所示。

```
this.mappedByteBuffer.force();
```

这个MappedByteBuffer就是JDK NIO包下的API，他的force()方法就是强迫把你写入内存的数据刷入到磁盘文件里去，到此就是同步刷盘成功了。

那么如果是异步刷盘呢？我们先看CommitLog.handleDiskFlush()里的的代码片段。

```
if (!this.defaultMessageStore.getMessageStoreConfig()
            .isTransientStorePoolEnable()) {
    flushCommitLogService.wakeup();
} else {
    commitLogService.wakeup();
}
```

其实这里就是唤醒了一个flushCommitLogService组件，那么他是什么呢？看下面的代码片段。

FlushCommitLogService其实是一个线程，他是个抽象父类，他的子类是CommitRealTimeService，所以真正唤醒的是他的子类代表的线程。

```
abstract class FlushCommitLogService extends ServiceThread {
  protected static final int RETRY_TIMES_OVER = 10;
}

class CommitRealTimeService extends FlushCommitLogService {

    public void run() {

    }

}
```

具体在子类线程的run()方法里就有定时刷新的逻辑，这里就不赘述了，这里留做大家的课下作业。

其实简单来说，就是每隔一定时间执行一次刷盘，最大间隔是10s，所以一旦执行异步刷盘，那么最多就是10秒就会执行一次刷盘。

好了，到此为止，我们把CommitLog的同步刷盘和异步刷盘两种策略的核心源码也讲解完了。我们主要是讲解的核心源码，而源码里很多细节不可能一行一行进行分析，大家可以顺着文中的思路继续探究。

End

重要说明：

- 如何提问：每篇文章都有评论区，大家可以尽情留言提问，我会逐一答疑
- 如何加群：购买狸猫技术窝专栏的小伙伴都可以加入狸猫技术交流群，一个非常纯粹的技术交流的地方

  具体加群方式，请参见目录菜单下的文档：《付费用户如何加群》（**购买后可见**）